N89-16328

A Software Development Environment Utilizing PAMELA

Prepared by:

R. L. Flick
&
R. W. Connelly

Westinghouse Defense & Electronics Center
P.O. Box 746
Baltimore, Maryland 21203

May 12, 1986

A Software Development Environment Utilizing PAMELA[1]

Abstract

Hardware capability and efficiency has increased dramatically since the invention of the computer, while software programmer productivity and efficiency has remained at a relatively low level. A user-friendly, adaptable, integrated software development environment is needed to alleviate this problem. The environment should be designed around the Ada[2] language and a design methodology which takes advantage of the features of the Ada language such as the Process Abstraction Method for Embedded Large Applications (PAMELA).

Introduction

Since the invention of the computer, advances in software development productivity have not kept pace with hardware productivity. Although the throughput of modern computers has made a 1,000,000-fold increase over the last thirty years, software productivity has increased only slightly. During the same period, hardware costs have decreased dramatically and software costs have skyrocketed. Moreover, the complexity of embedded systems is growing exponentially, putting an ever increasing demand on software production.

Many studies have shown that the major costs in the software development life cycle occur after system delivery. Approximately 70% of these costs are incurred during the maintenance phase. There are several reasons for this:

1. Personnel costs for software professionals have risen steadily over the years. Consequently, for large systems designed to last many years, the cost of people becomes a major concern.

2. Inadequacy of documentation either internal or external to the code is a continual source of increased costs. Frequently on large systems, a modification in one routine will affect many other routines in unexpected ways. It is not uncommon that a

1. PAMELA is a trademark of Dr. George Cherry, Reston, Virginia
2. Ada is a registered trademark of the United States Department of Defense, Ada Joint Program Office.

change to correct one error will lead to numerous other errors. This is partially because large programs are intrinsically hard to understand, but also because inadequate documentation hampers understanding. Furthermore, most programming languages do not promote greater understanding since they do not always enforce good software engineering practices.

3. Yet another factor is an inadequate design process. Frequently, paper designs are created by systems engineers and then handed to programmers for implementation. The programmer often will tend to stray from the paper design in order to increase efficiency or make changes that are required by the constraints of the language employed.

The real cost of software therefore is in the maintenance of programs - but it originates in the methods and languages used to create these programs.

Current projections show that the cost of developing software is likely to continue to increase unless new, more efficient methods are employed. If current trends continue there will be a short fall of programmers by 1990 which may exceed 800,000.[3] Such a devastating short fall will slow software development to a crawl for many major government programs.

Current trends can be reversed by developing and utilizing standard software engineering practices throughout the software industry. These practices can be implemented in an expert system that is designed to specifically support one design methodology. In addition the methodology used must be specific to the language that is supported. The preferred language to be used is Ada, and one methodology that is specifically designed for the Ada language is PAMELA.

The Ada Language

Since the software development environment supports development of large embedded applications for the Department Of Defense (DOD) applications, and incorporate state-of-the-art tools, the language of choice is Ada. Ada is a fairly new language developed by the DOD specifically for embedded applications. Although Ada is new, the DOD has set a requirement that all new software written for the DOD will be done in Ada. As the advantages of using Ada as a general purpose programming

3. Mr. Edward Berard, EVB Systems, ACM SIGAda meeting, Los Angeles, California, February, 1986.

language become more fully developed, commercial firms will also choose Ada for there software needs. Some of the important features brought to software engineering are :

Code Reusability:

Ada supports code reusability in the form of generic packages, a common library of compiled units, and modular coding techniques. With these facilities Westinghouse has established a common database of program modules at the company level. By establishing and using this database of reusable software modules, generating software for embedded applications has become cheaper and faster.

Tasking:

Something new that is supported by Ada and virtually no other language, is the task unit. This unit is on the same level as a function or subprogram with one important difference: a task unit can be declared as a type. Because of tasking, generating embedded systems that require some form of parallel processing is easier.

Parallel Development:

An important feature of Ada, is the ability to do parallel development. Ada offers this facility in the form of separate compilation units. Westinghouse has found that several individuals can work on different sections of the code and not interfere with each other, and that code development is not dependant on any special order of accomplishment (other than Ada's dependency rules of course). Westinghouse has been able to increase software engineering productivity by reducing the scope of dependencies within the software application.

Information Hiding:

Ada provides the facility to hide the underlying machine dependent representation of data items. This discourages the software engineer from depending on a machine specific characteristic when implementing a section of the software system. It also means that the code generated should be transportable to any other machine that supports a validated Ada compiler.

Strong Type Checking Across Separate Compilation Units:

Ada is a strongly typed language that will not allow nonconformant data types to be passed between program units. The purpose of Ada's strong type checking is to prevent common errors from occurring when calling another software engineer's code.

Ada's Place In The Design Of A Software System:

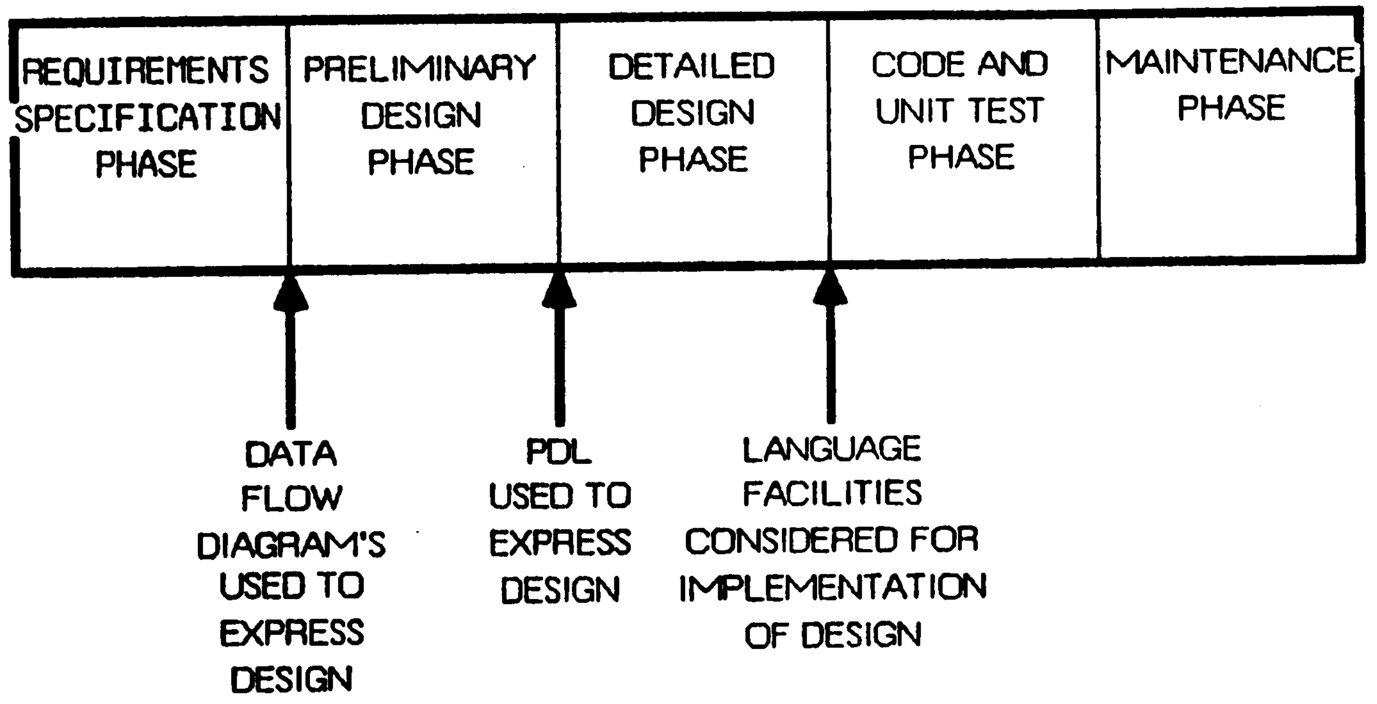
To be able to take advantage of the state-of-the-art facilities that Ada offers, the perception of when the capabilities of the programming language that is used is to be considered must be changed. In addition a design methodology that implements design concepts specified in MIL-STD 2167 and takes into account the improved facilities of Ada must be utilized.

Up until now the typical method for designing a software system involved specifying requirements, doing a preliminary design, doing a detailed design using a PDL and finally selecting a language and implementing the design. The primary methodology used when designing the system was typically a derivative of Data Flow Diagrams. (see Fig. # 1) This approach has worked with other languages (before Ada) because they did not provide sophisticated facilities for embedded environments such as tasking. Languages therefore, had little impact on the design of the system itself. All popular languages, aside from Ada, are sequential in nature. The design methodology used to express a system under development in this language is compatible with the capabilities of the language used and is sequential in nature.

If Ada is to become an effective alternative, several common practices and assumptions used in designing an embedded system must change, and a design methodology that is designed to accommodate a specific language must be used. To be able to take advantage of the advanced features that Ada offers, the methodology must take the language features into consideration in the preliminary design phase of a software system (see Fig. # 2). This means that consideration of language facilities should be an integral part of the preliminary and detailed design of the system. If the language considerations are made early in the development of the preliminary design, the overall impact will be in the areas of coding and integration time. These two areas comprise most of a software systems development cost. If however, Ada's facilities are not considered early in the preliminary development, Ada will offer almost no advantage over any other language.

The method of considering Ada's facilities in the preliminary and detailed design phase is dependent on the methodology used to express these designs. The popular methodologies of flow chart's, data flow diagrams, etc. will be of little advantage in the preliminary design phase when using Ada. The inability of these methodologies to express the unique facilities of tasking, code reusability, modular design, and parallel development diminish their usefulness for creating a design based on

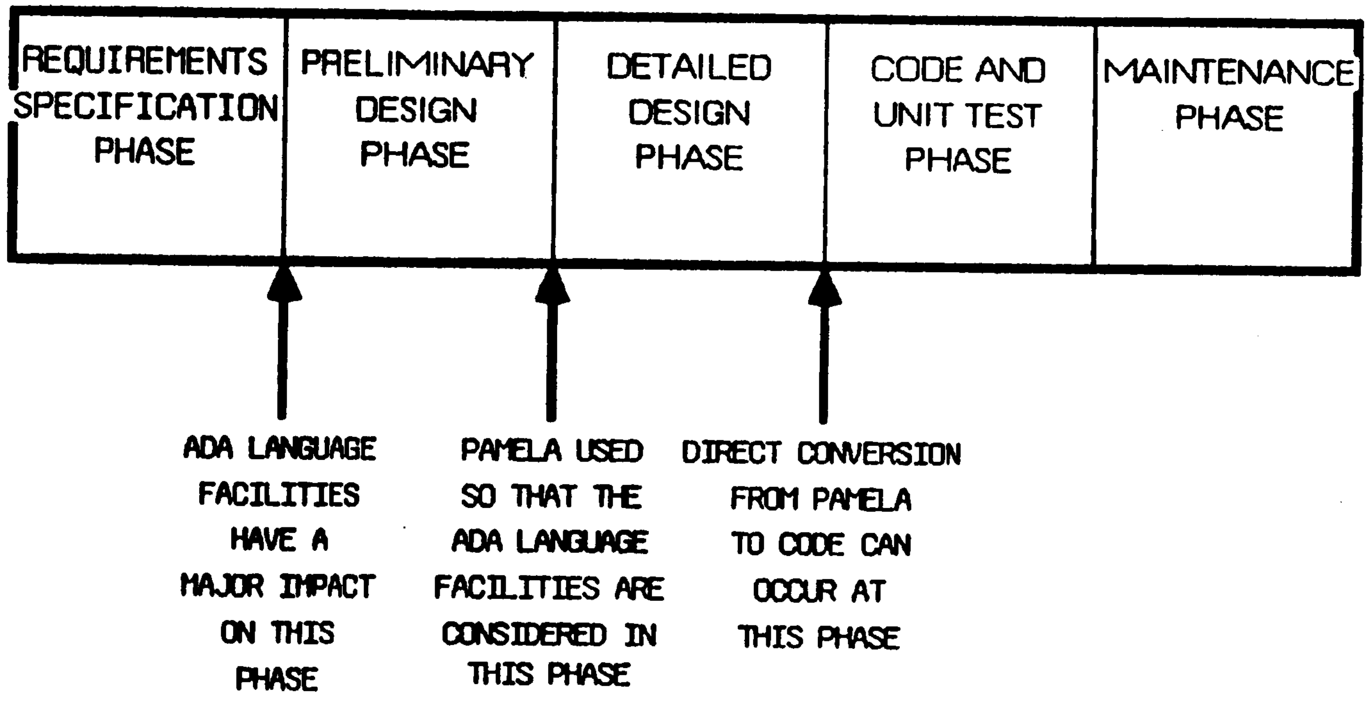
THIS METHODOLOGY REFLECTS THE SEQUENTIAL NATURE OF AVAILABLE LANGUAGES


REQUIREMENTS SPECIFICATION PHASE
PRELIMINARY DESIGN PHASE
DETAILED DESIGN PHASE
CODE AND UNIT TEST PHASE
MAINTENANCE PHASE
DATA FLOW DIAGRAM'S USED TO EXPRESS DESIGN
PDL USED TO EXPRESS DESIGN
LANGUAGE FACILITIES CONSIDERED FOR IMPLEMENTATION OF DESIGN

CURRENT METHOD OF SOFTWARE SYSTEM IMPLEMENTATION

FIGURE #1

THIS METHODOLOGY REFLECTS THE PARALLEL NATURE OF THE ADA LANGUAGE


REQUIREMENTS SPECIFICATION PHASE
PRELIMINARY DESIGN PHASE
DETAILED DESIGN PHASE
CODE AND UNIT TEST PHASE
MAINTENANCE PHASE
ADA LANGUAGE FACILITIES HAVE A MAJOR IMPACT ON THIS PHASE
PAMELA USED SO THAT THE ADA LANGUAGE FACILITIES ARE CONSIDERED IN THIS PHASE
DIRECT CONVERSION FROM PAMELA TO CODE CAN OCCUR AT THIS PHASE

SUGGESTED METHOD OF SOFTWARE SYSTEM IMPLEMENTATION

FIGURE #2

Ada. It is therefore necessary that a design methodology that can express parallelism, code reusability, modular design, and parallel development be used. It is also necessary that the methodology used in the preliminary and detailed design be a direct expression of the Ada language. We have found that PAMELA fits this description.

Description of PAMELA

PAMELA is a methodology for producing real-time Ada programs which utilize Ada tasking. It was designed by Dr. George Cherry (Reston, Virginia) to address the needs of Ada users in developing real-time programs using Ada's rich variety of language features.

PAMELA is a structured methodology that encourages a top-down approach, with each step in the method revealing more details than the previous step. (see Fig. # 3) It is also a graphical methodology which produces pictographs of the underlying Ada code. In fact, Ada package and task specifications, as well as skeletal package and task bodies, can be produced directly from PAMELA graphs.

PAMELA combines aspects the two most prominent program representation methodologies of the past two decades, data flow diagrams and control flow diagrams (flow charts). It's pictographs are very similar to those produced by structured analysis and structured design techniques (data flow diagrams), but it also embodies a certain amount of control flow information - primarily because of the well defined Ada tasking mechanism.

PAMELA guides the program designer in the selection of multiple, concurrent threads of execution (called **processes** in PAMELA nomenclature). By analyzing the requirements of the problem, and by following the process idioms outlined in the method, (see Fig. # 4) the program designer identifies which elements of the program should become processes. He then determines what kind of data or control signals must be passed between processes. Next, he determines which process is the producer of the flow and which is the consumer. (see Fig. # 5) Finally, he determines which of the processes should be single-thread (a typical C, PASCAL, or FORTRAN - style program) and which should be multi-thread (more than one Ada task). Once the graph has been annotated with this information, Ada code can automatically be generated (in skeletal form) which implements the design.

PAMELA

- Define External Entities
- Decompose Master Procedure Into Processes
- Map All I/O From Higher Level Graph to Decomposed Graph
- Identify Internal Interfaces Between Processes
- Identify Calling Process

FIGURE #3

DEFINITION OF A PROCESS

- Hardware — dependent, application — independent function
- Device driver
- Interrupt handler
- Buffer between asynchronous activities
- Application — dependent, hardware — independent function
- Proprietor (monitor)
- Cyclic activity
- Activity requiring a distinct priority
- Vigilant activity
- Activity that must wait a specified time for an event

Reference: Process Abstraction Method for Embedded Large Applications
George W. Cherry, Reston VA 1985.

FIGURE #4

LEVEL 1

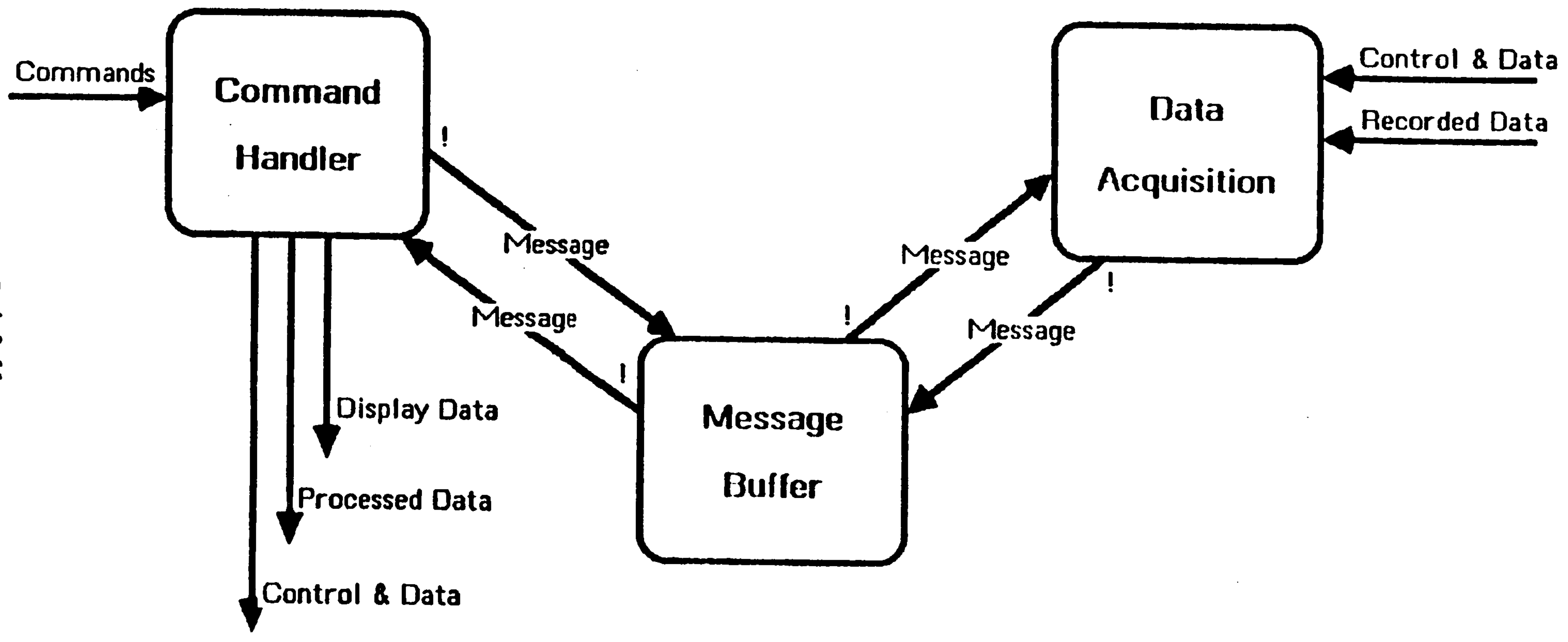


FIGURE #5

From the PAMELA graphs, single-thread processes become Ada tasks, while multi-thread processes become packages. The package body of a multi-thread process contains task and package specifications for the lower level single- and multi-thread processes respectively.

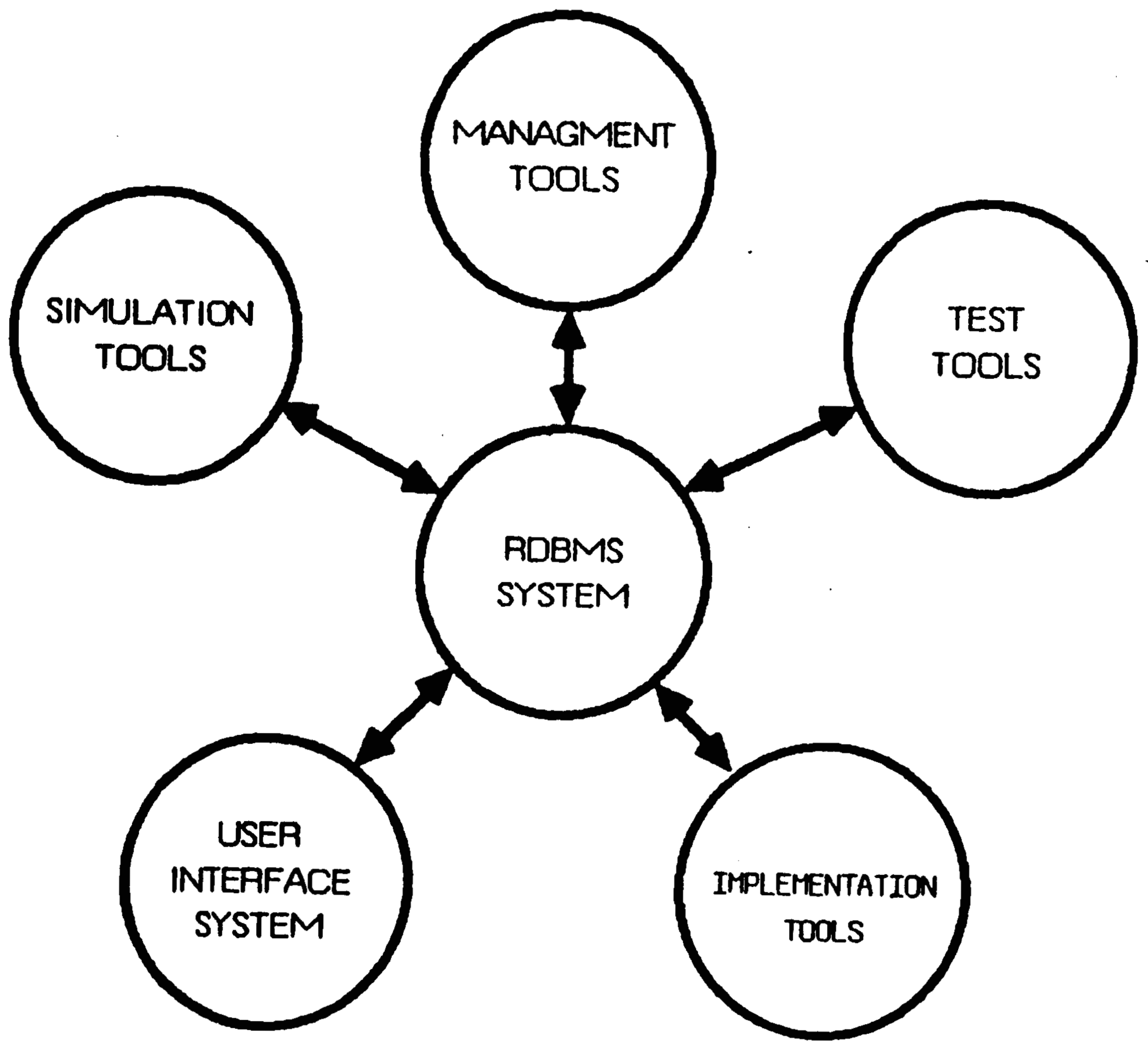
The Problem To Be Solved

The problem of increasingly large and complex software systems, in concert with a massive projected shortfall of software engineers in the next decade, fueled by spiraling software costs, must be abated. It is foolhardy to think that software systems will decrease in complexity; all current trends support the notion that future software systems will be very much more complex than those of today. The number of software engineers may increase by the next decade, but probably not fast enough to meet the challenges of these more complex systems. If software engineers continue to be in high demand, there is little hope of abating spiraling software costs.

The key to the solution of the problem is to substantially increase the productivity of software professionals. The primary tool to accomplish this goal is a high performance Software Development Environment (SDE). The SDE must be designed and built around a single specific language and design methodology. Since the DOD has mandated that all new software written for the DOD will be in Ada, Ada is the natural language of choice for the SDE. There are several new Ada based design methodologies such as Object Oriented Design (OOD), PAMELA (Process Abstraction Methodology for Embedded Large Applications), and Ada Partition Programming Language (APPL). Of all the new design methodologies we are considering PAMELA, as an example, around which to design the SDE.

The Software Development Environment

A software development environment (SDE) is being created at Westinghouse which supports all activities associated with the development of embedded software systems, as well as software management and post deployment support. By integrating all of the activities involved in software development under the control of one expandable, adaptable environment, software development and support can be made easier, more cost effective, and more reliable (see Fig. # 6).



SOFTWARE DEVELOPMENT ENVIRONMENT SYSTEM DIAGRAM

FIGURE #6

Important elements of the environment are:

Reliability:

The reliability of programs created under the SDE must be significantly greater than that of programs generated without such an environment. Reliability metrics, when applied to programs created under the SDE, should show a measurable and statistically significant increase in reliability. This in turn will require that the SDE itself be an exceptionally reliable program. We have seen that by using PAMELA, it is relatively straightforward to create reliable designs in a timely manner. Since the underlying Ada code maps directly to PAMELA pictographs, it is only necessary to correctly identify control and data flows at a high (pictograph) level to insure the reliability of the underlying code. Application of expert system techniques will also enhance the reliability of the environment.

Ease of Use:

The SDE will encompass a common, multi-level, user friendly interface. In particular, the interface will be as easy to use for the novice as for the expert. This will probably be accomplished with a multi-window, menu-driven interface which will provide full prompting for the novice. For the expert, a series of function keys and/or control keys can be defined (by the environment and/or by the user) to enable rapid execution of frequently used command sequences. For others, on-line help and an English-like command interface will be provided. Every user will be able to select the interface he/she prefers and will also be allowed to jump to any particular interface level at will.

PAMELA will support the ease of use concept since it is graphical, and is supported by an interactive, full screen tool which can automatically generate executable code.

Cost Effectiveness:

The environment should be networked so that individual workstations can be utilized by development and management personnel. This means that each individual or team will be able to achieve maximum utilization of the facilities available while avoiding the typical slow down experienced with multi-user super-mini implementations. Because of advances in micro-processors, a single user workstation can provide an engineer with a more responsive machine than can normally be attained with a time-shared super-mini. The resultant increase in throughput, can increase productivity substantially. As a side benefit, costs incurred due to main CPU down-time can be minimized by allowing the workstations to operate independent from the host.

The use of PAMELA should also prove cost effective in that it allows for rapid prototyping of the software system within the SDE. This allows the program implementors (and designers) to identify and correct potential or unexpected problem areas before they actually become problem areas.

Adaptability:

The environment will support various tools that will measure productivity, quality, maintainability and overall cost. This means that management will have the ability to measure all aspects of the evolving system in terms of quality, maintainability and cost. It will also allow the measurement of team performance compared to calibration data contained in the database. Such measurements can be used to recalibrate the system to more accurately reflect real world situations.

PAMELA has proven itself to be quite adaptable. In one particular instance, a 7000 line program was re-designed and re-implemented from scratch in just three days.

Design Continuity:

The environment is an expert system which provides tools that enhance all phases of the software life cycle. Program requirements are entered into a relational data base under the control of the expert system. Once a requirement has been entered, a basis is established for all later phases of the software life cycle. In particular, design, coding, and test specifications are derived from the requirements and related back to them by the expert system. This provides traceability from requirements to code, but also allows the environment to provide an impact analysis report for each requirement.

Software designs (specifically PAMELA designs) are accepted by the expert system. Once a design has been entered, it can be verified for compliance to the requirements by the environment.

Tools such as language sensitive editors, compilers, and debuggers which facilitate the coding and unit testing process can also be directed by the environment. For example, a compiler which produces diagnostic information could relate the number and kinds of programmer errors to the environment. The expert system could in turn relate this information back to a language sensitive editor to help correct programmer mistakes as they happen.

Since all requirements and design information are entered into the expert environment, test scenarios and/or test cases can be automatically generated to verify the design.

Program Visibility

The SDE supports all levels of management visibility into the current status, and projected results of the project. This means that the management functions of progress tracking, scheduling, and cost information gathering will be provided by the expert system. This includes but is not limited to, the automatic generation documentation and management reports with little or no human intervention. In addition there is some capability of the software factory concept in that generic, reusable software could be placed in the design by the expert system itself. This will alleviate the problem of the software engineer overlooking a reusable package that is in the database of reusable program modules.

Projected Environment Layout:

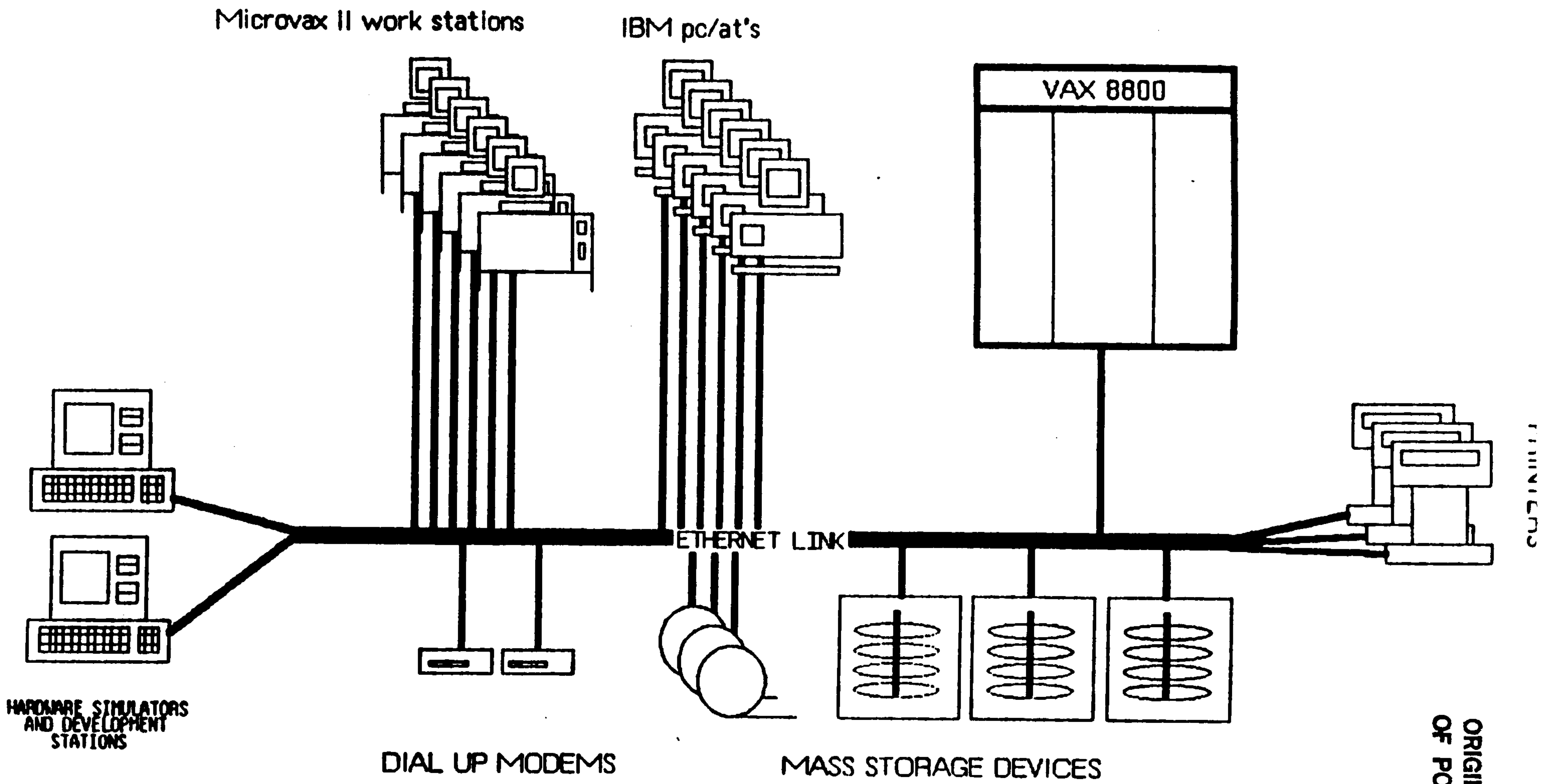
The environment will be as flexible as possible and support all types and sizes of software development. The system will incorporate artificial intelligence, networking, database management and some form of electronic mail. The hardware of such a system is projected to be composed of the following components:

1. A VAX minicomputer as the central database machine.
2. Several VAXSTATION II's as individual workstations.
3. Several micro computers such as IBM PC/AT's for manager workstations.
4. Some type of clustering system.
5. Some type of LAN (Local Area Network) system for node communications.
6. Hardware simulators and development stations for hardware specific support.

(see Fig. # 7)

Potential Problem Arenas

As is the case with all things, PAMELA is not perfect. There are two potential problem arenas associated with PAMELA which affect the performance of the SDE. For one thing, PAMELA designs typically


Microvax II work stations
IBM pc/at's
VAX 8800
ETHERNET LINK
HARDWARE SIMULATORS AND DEVELOPMENT STATIONS
DIAL UP MODEMS
MASS STORAGE DEVICES

ORIGINAL PAGE IS
OF POOR QUALITY

THE SOFTWARE DEVELOPMENT ENVIROMENT HARDWARE ARRANGEMENT

FIGURE #7

create too many tasks. This is not a fault with the methodology per se, but reflects the fact that there are precious few machines out there that are made to run Ada. The methodology has been altered somewhat to account for this fact, but in so doing, it has lost some of its "virtual machine" flavor.

Another potential problem arena is that of testing. The current suggestion is to test each single-thread process using current structured techniques. As each is tested, it is integrated with the others and an integration test is performed. Eventually a multi-thread process will be declared valid and it then can be integrated with other processes. There is no method however for verifying that all the task rendezvous and other task interactions are correct. This is still a matter of art as much as it is of science but may be alleviated somewhat by the use of heuristic approaches common in expert systems. It is not clear however, whether this will be harmful for large embedded systems. If the paper design is solid, the implementation should be as well; but there is unfortunately no method for verifying paper designs either.

Potential Solutions

The horizon should not be clouded by the concerns raised above. Each problem poses new and exciting possibilities for new technologies and new ideas to solve those problems. Each new challenge brings us closer yet to another breakthrough.

The problems posed in the development of a state-of-the-art software development environment can be solved by hard work and dedication. They should not be attacked alone, but in concert with concerned organizations willing to lead us into the next century.

Conclusions

In conclusion, the need for a comprehensive, integrated software development environment has been demonstrated by the severe lack of productivity in developing software as compared to computer hardware. The need to automate documentation so that it provides a better picture of the program is essential to decreasing the maintenance costs of large software systems. An automated, integrated environment supporting a single specific language such as Ada and designed around a specific methodology such as PAMELA will reduce time and errors in the design and testing phases. Since the environment will ensure adequate tracking of requirements, design, implementation and testing, the cohesion is provided to aid management tracking of progress during the software life cycle. A common, multi-level, user friendly interface is absolutely required to insure maximum effectiveness for all users of various levels of experience and expertise. Finally, PAMELA is an ideal design methodology for such an environment, since it is Ada-based, and naturally addresses multiple concurrent tasks. PAMELA has been used on projects at Westinghouse and has proven its effectiveness for rapid prototyping, ease of design, maintainability and adaptability.[4]

4. Some material contained in this document was presented by Rich Connelly and Barbara Sullivan at the SigAda conference held in Boston Ma. in Nov. 1985